PYTHON OOP – ITS ALL PREDICATE LOGIC

Tools:

- Doxygen

0. Ultimately all names references maps to executable code somewhere in memory.

1. Understand the process of class creation – from using metaclasses.

- Abstract classes allows the blueprient of classes
- Classes allow the blueprints of objects – its hierachy
- We have inheritance, encapsulation and polymophism
- Undestand class descriptors – for validation and controlling attributes access: useful in large codebase to have one class to customise others; else getter and setter or simple try and except can do the job.
- Class special methods to override and called by builtin functions.
- A class namespace (a dictionary which binds class names to their object in memory or its file)
- Undesrstand class methods achieved throught decorators - @classmethod and @staticmethod
- Using classes as functions through __call__ method.
- self and cls are very important, they must be provided on methods since methods must act on the right object (a pointer to the object on the stack)
- __init__(): init and its arguments are the class's signature. __new__ accepts *args and **kwargs to pass to __init__, but __init__ must have proper arguments which influence __new__ args and kwargs.

2. A python script has the following sections – code blocks:

- import systems creates a name space that includes all the objects defined in the module, use __dict__ or __dir__.
- Classes and functions create their own namespaces
- We have the all important builtin namespace – links to C code and compiled into the interpreter.
- Use __dict__, __dir__, globals, and locals to investigate objects that one can use.

3. The CPython interpreter and GCC compiler:

- its a stack based machine – it is a data structure which executes python .pyc file
- it is created by a normal C function during initialisation, Cpython has its stack, heap and code sections just like any normal C code, and it operates just like any normal C code.
- Python source code from a file is read like a stream of bytes into Cpython address space (where?), and then evaluated to produce a .pyc file (stored on disk).
- Like gcc it works as the above line – but it creates an independent .so or .o file that can run on its own, loaded in memory independently of gcc.

4. Relationships that can be modelled:

- is-a-type-of relationship – normal inheritance models such, the parent class are most general and when travesing down to the bottom classes are becoming specialities. Problem is that it exposes even methods and attributes that are not necessary.

- has-a relationship – building an object out of components, its a class, like a body class, that is assembled from class components, like hands and legs. Its a class built out of many classes froming a component diagram.
- Closely related to has-a relationship is dependency injection – components are passed to the body class constructor, like components can be injected out or put back it ducing instatiation.
- Delation relationship – delagation isclosely related to decorators or fucntions that define inner objects and return those inner objects waiting to be executed. Delagation can be achived using the __getattr__ method called from some class to access methods from another class, useful when dealing with Mixin classes.

4. Abstarct base classes

- Provide a consistent API among different objects thereby promoting polymophism.

5. Class protocols - they provide special methods that are automatically triggered under certain situations to provide special functionalities – general functionalities per se.s

- descriptor protocol – managing attributes, just like @property decorators
- iterable protocol – name says, makes you class iterable
- iterator protocol – name syas, makes you class an iterator
- context mangers protocol – makes your classes to be used with context manager with syntax